Languages and P Systems: Recent Developments

Gheorghe Păun[1,2], Mario J. Pérez-Jiménez[2]

[1] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania

[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, marper@us.es

Summary. Languages appeared from the very beginning in membrane computing, by their length sets or directly as sets of strings. We briefly recall here this relationship, with some details about certain recent developments. In particular, we discuss the possibility to associate a control word with a computation in a P system. An improvement of a result concerning the control words of spiking neural P systems is given: regular languages can be obtained as control words of such systems with only four neurons (and with usual extended rules: no more spikes are produces than consumed). Several research topics are pointed out.

1 Introduction

Basically, membrane computing is associated with multiset processing in the compartments defined by a membrane structure, hence with handling *numbers* encoded in a *unary* manner, by means of the multiplicity of given objects, represented by symbols of an alphabet. However, from the very beginning, [22], also P systems were considered whose objects are strings. While the multisets of objects are processed by biochemical or biological inspired rules (similar to reactions taking place among the chemicals in a cell, or by other operations, such as symport and antiport), the string objects should be processed by specific rules, such as rewriting, splicing (from DNA computing), replication. However, also in the case of symbol objects we can "compute" (generate, accept or translate) strings and languages, and we find this case particularly interesting, taking into account the qualitative difference between the "internal data structure", the multiset, and the "external data structure", the string (hence with a positional information). That is why in what follows we only discuss this case, of symbol objects P systems handling languages.

For P systems with string objects we refer to the corresponding chapter of [31] and to the current bibliography of membrane computing from [38]. It is important

to note, however, that P systems with string objects can have interesting applications in natural language processing; we refer only to [1], but researches of the same group should be followed in this respect.

In what follows, we assume that the reader is familiar with basic facts in membrane computing, including definitions of the main classes of P systems: cell-like P systems with symbol objects (called here *transition P systems*, P systems with active membranes, symport-antiport P systems, spiking neural P systems (in short, SN P systems). Details can be found in [23], [31], and at [38]. We also assume some familiarity with basic elements of formal language theory, e.g., from [34]. Some notations will be also given below; we only mention now that REG, LIN, CF, CS, RE denote the families of regular, linear, context-free, context-sensitive, and recursively enumerable languages, respectively, and that V^* is the set of all strings over the alphabet V, the empty string, λ, included.

Informal presentations of the four classes of P systems are given below, in order to facilitate the understanding of the subsequent sections.

A *transition P system* uses rules of the form $u \to v$, where u and v are strings over a given alphabet O of objects, representing multisets; the intuition is that the objects in the multiset (represented by) u are consumed and those in v are produced, like in a (bio)chemical reaction. The objects in v can have associated *target indications*, in the forms $(a, here), (a, in), (a, out)$; the meaning is that the object a produced by applying the rule remains in the same compartment of the membrane structure if *here* is associated with it, it goes to a membrane immediately inside the compartment where the rule is used, or it goes outside this compartment, in the surrounding compartment, if the indications *in* or *out* are associated, respectively. Note that the objects are processed inside compartments, by local rules, but they can travel across membranes, due to the target indications. In particular, an object (a, out) produced in the external membrane of a P system (also called skin membrane) leaves the system and it "gets lost" in the environment.

Rules of the general form $u \to v$ are called *cooperative*. If u consists of a single object, then the rule is said to be *non-cooperative*. The intermediate case of rules $ca \to cv$, where a and c are objects, with c taken from a distinguished subset C of O, is the *catalytic* case.

In *P systems with active membranes*, the membranes themselves are part of rules and can evolve during a computation. The objects can evolve inside compartments (by cooperative, catalytic or non-cooperative rules) and can pass across membranes, while membranes can get divided, dissolved, separated, etc.

In P systems with symport-antiport rules the objects pass across membranes by rules of the forms $(u, in), (u, out)$ (symport rules), and $(u, out; v, in)$ (antiport rules), where u, v are strings in O^* (representing multisets of objects). The rules are associated with the membranes, the objects are never modified, they are just moved from a compartment to another one.

Starting from an initial configuration (the membrane membrane structure and the multisets placed in its compartments), and using the rules in a specified way

(synchronously or unsynchronously, in the maximally parallel way, sequentially, etc.), we get *transitions* among configurations; a sequence of transitions forms a *computation*; a computation which reaches a configuration where no rule can be applied is said to be *halting*. In all the previous cases, the most natural result of a computation is a number, for instance, of objects present in the halting configuration in a specified membrane.

In what follows, always the P systems work in the maximally parallel manner.

Finally, *an SN P system* consists of a set of *neurons* placed in the nodes of a directed graph and sending signals (*spikes*, denoted in what follows by the symbol a) along *synapses* (arcs of the graph). The objects evolve by means of *spiking rules*, which are of the form $E/a^c \to a; d$, where E is a regular expression over $\{a\}$ and c, d are natural numbers, $c \geq 1, d \geq 0$. The meaning is that a neuron containing k spikes such that $a^k \in L(E), k \geq c$, can consume c spikes and produce one spike, after a delay of d steps. This spike is sent to all neurons to which a synapse exists outgoing from the neuron where the rule was applied. There also are *forgetting rules*, of the form $a^s \to \lambda$, with the meaning that $s \geq 1$ spikes are forgotten, provided that the neuron contains exactly s spikes. If rules can produce more than one spike, i.e., they are of the form $E/a^c \to a^p; d$, with E, c, d as above and $1 \leq p \leq c$, then the system is said to be *extended*. (Note that the number p of produced spikes cannot be greater than the number c of consumed spikes.) In the initial configuration, each neuron contains a given number (it can be zero) of spikes.

The system works in a synchronized manner, i.e., in each time unit, the rule to be applied in each neuron is non-deterministically chosen, each neuron which can use a rule should do it, but the work of the system is sequential in each neuron: only (at most) one rule is used in each neuron. One of the neurons is considered to be the *output neuron*, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* of the system – it might be infinite if the computation does not stop. The *result of a computation* can be the spike train itself (a binary string if the computation halts, or an infinite sequence otherwise) or a number (e.g., the distance between the first two spikes sent into the environment by the output neuron of the system).

If a spiking rule $E/a^c \to a * p$ has $L(E) = a^c$, then we write it in the simpler form $a^c \to a^p$ (and we call it *finite*).

Four ways to associate a language with a P system were considered so far:

1. external output,
2. using a P system in the accepting mode,
3. following the trace of a distinguished object through the membrane structure,
4. control words.

We shortly present them below, with some details in the case of control words, and then we propose some ideas for further research.

It should be noted that the references we give here are not meant to be complete or to indicate the first place where a notion was introduced, but only to offer a good introduction to this research area.

A *general research topic* can already be formulated here: consider systematically the 4×4 combinations of (basic) types of P systems and ways to associate a language with a P system. Not all of these 16 possibilities were explored (but we cannot say in advance that any of them is of no interest). In particular, equivalences between some of the 16 combinations would be nice to be found.

2 External Output

Introduced already in [30], for transition P systems, the idea is simple: because objects can exit a P system (of any type), we (the user, the observer) can "wait in the environment" and arrange the symbols which leave the system in a sequence. If the computation halts, then we obtain a string, if not, we obtain an infinite sequence. An important detail: we have to decide what to do in the case when several objects leave the system at the same time. In [30] and several subsequent papers, all permutations of the symbols are allowed, hence several strings are associated with the same computation. An interesting possibility is to disregard certain symbols and/or to associate a single symbol to a multiset (by means of a given "interpretation mapping"), like in [12].

Somewhat surprisingly, in spite of its simple definition, defining a language in the external output manner was not too much investigated – at least not until last years, when a systematic study was started in [3], [4], mainly for transition P systems with non-cooperative rules (and no further ingredients; in [30], catalytic rules and membrane dissolution rules are used, as well a priority relation among them). The obtained family lies in between REG and CS and has interesting (combinatorial) properties.

The spike train of an SN P system can also be considered as the result of a computation defined in the external mode, but, having only one object, we have to assign different symbols to the time units when (at least) a spike exits the system and to the time units when no spike is emitted. In this way, a binary string (or sequence, when the computation does not stop) is obtained. There are several papers in the SN P systems area dealing with such languages.

The external output is not very much investigated for symport-antiport systems, and we know no paper of this kind dealing with P systems with active membranes. Also, as far as we know, the case when only computations which send out at most one object in each step was not investigated (this condition imposes a restriction on the accepted computations, hence the computing power of P systems can be altered in this way).

3 P Automata

This is indeed a much investigated topic in membrane computing – but mainly for the symport-antiport case. The idea is simple (symmetric to the external output): we arrange in a sequence the symbols which enter a P system, again with the two possibilities, to consider all permutations of symbols which enter in the same step (see [21] and its bibliography), or to consider an encoding of multisets by symbols (see a survey and references in [12]).

For symport-antiport P systems the "reading" of symbols from the environment is naturally defined by means of symport and antiport rules associated with the skin membrane. This is also provided by rules with active membranes, but we know no study about this issue for such systems. For transition P systems and for SN P systems we have to input symbols in an "external manner" (an external user provides a string, symbol by symbol, according to its wish). In most cases, a string is accepted if the computation halts (there are also other ways to define successful computations, such as local halting, reaching final configurations, but we do not discuss them here).

This way of using P systems is also related to the use of P systems to solve decidability problems, were an input is introduced in the system and the problem (an instance of it) has a positive answer if the computation halts (and a special object is sent to the environment), however, in this case the input (an encoding of the instance of the problem) is introduced in the form of a multiset, placed in a distinguished membrane. Details can be found in [32].

A recent variant of P automata was introduced in [26], called *dP automata*: several (symport-antiport) systems are connected to each other by means of antiport-like rules; they read separately strings from the environment, process them, also communicating, and if the computation halts, then the concatenation of the input strings is accepted. This is a way to introduce more distribution in P systems, making explicit the splitting of a problem among the components of the dP automaton. There are several papers devoted to this topic, see, e.g., [27], [28], [37]. The idea was extended also to SN P systems, in [19]; in this context, also a dual of spiking rules is introduced, in the form of *request rules* (depending on the contents of a neuron, spikes can be brought in from the environment, that is, the spikes come in *by request*, not introduced by an external user).

4 Traces

The idea was introduced in [18] for symport-antiport P systems, investigated in a couple of papers (see [17] and its bibliography), and extended to SN P systems in [7]: distinguish an object and follow its path across membranes; the sequence of membrane labels visited by that object provides a string (in the case of SN P systems, one single spike is distinguished, it it always used by a spiking rule applied in the neuron where the marked spike resides and one of the produced

spikes become marked). We know no paper dealing with traces in transition P systems and in P systems with active membranes.

5 Control Words

Finally, the fourth way to associate a string with a computation is to consider *control words*, as sequences of labels of rules used in the steps of a computation.

This is a well investigated topic in formal language theory, especially for Chomsky grammars, because in each step such grammars use only one rule. Each derivation produces a control word; the set of all control words associated with all terminal derivations in a grammar is called the *Szilard* language associated with (generated by) the grammar. The things become more complicated in the case of parallel computing devices, when several rules are used simultaneously.

This is the case also in membrane computing, and probably this is the reason why control words were, up to our knowledge, never considered in this area (until the special case proposed in [33]). However, a sort of bidimensional control word was introduced already in [10], under the name of *Sevilla carpet*, as a way to describe the rules used in a computation and their multiplicity in each step, but not as a way to define a control language associated with the computations in a P system.

A possible solution to the above difficulty is to consider a sequence of multisets of labels, those labels associated with all rules applied in a given step. Then, a string of symbols can be obtained following the ideas also used for accepting P systems: take a function from multisets to strings and build the string(s) obtained by concatenating the strings associated with the multisets. For instance, all permutations of the labels in a multiset can be considered, as in [21], or only one specific string (maybe a symbol) associated with the multiset, like in [12].

Another idea was recently introduced in [33], starting from the following restriction: all rules used in a computation step should have the same label, or they can also be labeled with λ.

The definition in [33] is given for SN P systems, but it works for any type of P systems.

Indeed, let us consider a P system Π, of any type, with the total set of rules (the union of all sets of rules associated with compartments, membranes, neurons – as it is the case) denoted with R. Consider a labeling mapping $l : R \to B \cup \{\lambda\}$, where B is an alphabet. We consider only transitions $s \Longrightarrow_b s'$, between configurations s, s' of Π, which use only rules with the same label b and rules labeled with λ. We say that such a transition is *label restricted*. With a label restricted transition we associate the symbol b if at least one rule with label b is used; if all used rules have the label λ, then we associate λ to this transition. Thus, with any computation in Π starting from the initial configuration and proceeding through label restricted transitions we associate a (control) word. The language of control words associated with all label restricted halting computations in Π is denoted

by $Sz_\lambda(\Pi)$. The subscript indicates the fact that λ steps are permitted; in the opposite case, we write $Sz(\Pi)$ (the label restricted transitions which cannot use only rules with label λ are called λ-*label restricted*).

We give here two results for symport-antiport P systems. The family of languages $Sz(\Pi)$ associated with symport-antiport P systems with at most m membranes is denoted with $SzSAP_m$; when λ moves are allowed, we write $Sz_\lambda SAP_m$, and if the number of membranes is not bounded, then the subscript m is replaced with $*$.

In what follows we need the characterizations of regular languages by means of *regular grammars*. Such a device is a construct $G = (N, T, S, P)$, where N, T are disjoint alphabets (the nonterminal and the terminal one, respectively), $S \in N$ (the axiom), and P is a finite set of rewriting rules of the forms $A \to aB, A \to a$, where $A, B \in N$ and $a \in T$; a rule $S \to \lambda$ can be added, if we also want to generated the empty word. The language generated by G is denoted with $L(G)$. Without any loss of generality we may assume that the grammar is *reduced*: each $A \in N$ can be reached from S and can derive a terminal string.

When comparing two language generating or accepting devices G_1, G_2, the empty string is ignored, that is, $L(G_1)$ is considered equal to $L(G_2)$ as soon as $L(G_1) - \{\lambda\} = L(G_2) - \{\lambda\}$. Thus, no λ-rule is necessary in our regular grammars.

Theorem 1. $REG \subset SzSAP_1$.

Proof. The inclusion is easy to prove: for a regular grammar $G = (N, T, S, P)$ with $N = \{A_1 = S, A_2, \ldots, A_n\}$, we consider the antiport rules $b : (A_i, out; A_j, in)$ associated with $A_i \to bA_j \in P$ and the symport rules $b : (A_i, out)$ associated with $A_i \to b \in P$. Initially, the single membrane of the system contains the object A_1. Clearly, each terminal derivation in G corresponds to a halting computation in the system we have constructed, and conversely.

The inclusion is strict; actually, we have a stronger result: $SzSAP_1 - CF \neq \emptyset$. A P system proving this assertion is

$$\begin{aligned}
\Pi &= (O, [\ \ ]_1, e, O, R_1), \text{ where:}\\
O &= \{a_1, a_2, e, f, g, h\},\\
R_1 &= \{a : (e, out; ea_1a_2, in),\ a : (e, out; fa_1a_2, in),\\
&\quad b : (fa_1, out; f, in),\ b : (fa_1, out; g, in),\\
&\quad c : (ga_2, out; g, in),\ c : (ga_1, out; h, in),\\
&\quad d : (ha_1, out; ha_1, in),\ d : (ha_2, out; ha_2, in)\}.
\end{aligned}$$

The "carrier" e bring inside $n \geq 1$ copies of a_1 and a_2, then f and g remove copies of a_1 and a_2, respectively. Eventually, the object h is introduced in the system. If any copy of a_1 or a_2 is still present in the system, then the computation never halts, because the rules with label d can be used forever. Therefore, the control words associated with terminal computations are of the form $a^n b^n c^n$, for some $n \geq 1$, hence $Sz(\Pi)$ is not context-free. □

If steps when only rules with label λ are allowed, then all one-letter recursively enumerable languages can be generated.

Theorem 2. *If $L \subseteq a^*, L \in RE$, then $L \in Sz_\lambda SAP_1$.*

Proof. A language $L \in a^*$ is in RE if and only if its length set is a recursively enumerable set of numbers. Symport-antiport P systems with one membrane (and rules with no restricted complexity) can generate all recursively enumerable sets of numbers, [31]. Take such a system Π, namely, one which simulates a register machine $M = (n, H, l_0, l_h, I)$ (the number of registers, the set of instruction labels, the label of the initial instruction, the label of the halt instruction, the set of instructions, labeled with elements of H; simulating register machines is the usual way to prove the universality of symport-antiport P systems, so the reader is assumed to be familiar with such proofs). In the halting configuration, the system contains k copies of a symbol a_1, which encodes the contents of register 1 of M, the one where the number is generated, as well as the object l_h, for $k \in N(M)$. Assume that all rules of Π are labeled with λ, and add the following rules $a : (l_h a_1, out; l_h, in)$. This rule must be used for each copy of a_1 present in the system, hence the control word of the computation in the augmented system – let us denote it by Π' – is a^k. The halting label l_h is introduced only in the last step of a computation in Π. Consequently, $L = Sz_\lambda(\Pi')$. □

In the previous results we have imposed no restriction on the length of the symport and antiport rules; if such restriction are considered, then a larger number of membranes is expected to be necessary.

The control words associated with transition P systems and with systems with active membranes remain to be investigated. In what follows we consider the case of SN P systems.

6 Control Words for SN P Systems

The fact that λ steps increase the power of systems is also confirmed for the control words associated with SN P systems, a case which is investigated in [33]. Let $SzSNP_m$, $Sz_\lambda SNP_m$ be the families of all languages $Sz(\Pi)$, $Sz_\lambda(\Pi)$, respectively, associated with SN P systems Π (with extended rules) with at most m neurons; if the number of neurons is not restricted, then we replace the subscript m by $*$. In [33] it is proved that $Sz_\lambda SNP_* = RE$, but $SzSNP_* \subset CS$, strict inclusion (an example of a language not in $SzSNP_*$ is the linear language $\{xx^R \mid x \in V^*\}$, where V is an alphabet with at least two symbols and x^R is the reversal/mirror image of the string x). Moreover, a theorem in given in [33] stating that each regular languages L is the λ-label restricted Szilard language of an SN P system Π – with the mentioning that the system Π uses extended rules of the form $E/a^c \to a^p$ without the restriction $p \leq c$ and it has arbitrarily many neurons. This result will be improved in the next theorem.

We give first an example, also improving a result from [33], where it is shown that $SzSNP_6$ contains non-context-free languages. We prove that four neurons suffice.

Consider the SN P system (with four neurons, $\sigma_1, \sigma_2, \sigma_3, \sigma_4$)

$$\begin{aligned}
\Pi &= (\{a\}, \sigma_1, \sigma_2, \sigma_3, \sigma_4, syn), \text{ where:}\\
\sigma_1 &= \sigma_2 = (2, \{r_1 : a^2 \to a^2,\ r_2 : a^2 \to a\}),\\
\sigma_3 &= (1, \{r_2 : (a^4)^+ a/a \to a,\ r_3 : (a^4)^+ a^2/a^4 \to a\},\\
\sigma_4 &= (1, \{r_2 : (a^4)^+ a/a \to a,\ r_4 : (a^4)^+ a^2/a^4 \to a\},\\
syn &= \{(1,2), (2,1), (1,3), (1,4), (2,3), (2,4)\}.
\end{aligned}$$

The system is given in a graphical form in Figure 1. Each neuron contains initially one or two spikes, but only σ_1 and σ_2 can fire. If the rules r_2 are used in σ_1 and σ_2 (not also in σ_3, σ_4, because we do not have here enough spikes), then the computation halts. Let us assume that for a number n of steps we use the rule r_1 in σ_1 and σ_2. Neurons 1 and 2 exchange spikes to each other and, together, they send four spikes to each of σ_3, σ_4. These neurons cannot use the rules r_3, r_4 until getting inside an even number of spikes, and this means that the rules r_2 in σ_3, σ_4 were used. This however supposes that also σ_1, σ_2 use the rules r_2 (these rules are applicable, hence they must be applied), and this ends the work of these neurons. After using the rules r_2, neurons 3 and 4 can fire nondeterministically, but not both at the same time: they have to use the rules r_3 and r_4, which have different labels. After using the rules r_2, each of σ_3 and σ_4 contains the same number of spikes, namely $4n + 2$, hence, besides the string r_2, $Sz(\Pi)$ contains strings of the form $r_1^n r_2 w$, with $w \in \{r_3, r_4\}^*$ containing the same number of r_3 and r_4. This language is not context-free, hence $SzSNP_4 - CF \neq \emptyset$.

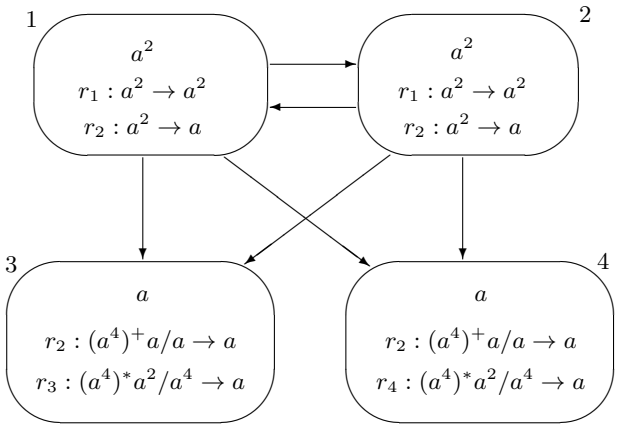

Fig. 1. An SN P system whose Szilard language is not context-free.

We give now the improvement of the mentioned result from [33].

Theorem 3. $REG \subset SzSNP_4$.

Proof. In view of the previous example, it is enough to prove the inclusion $REG \subseteq SzSNP$. To this aim, let us consider a regular language L generated by a reduced regular grammar $G = (N, T, S, P)$ with $N = \{S = A_1, A_2, \ldots, A_n\}$ and the rules in P of the forms $A_i \to bA_j$, $A_i \to b$, for some $A_i, A_j \in N$ and $b \in T$. Let us denote $J = \{1, 2, \ldots, n\}$.

We construct the following SN P system of degree 4 (together with the rules we also specify their labels):

$$\begin{aligned}
\Pi &= (\{a\}, (a^{2n+1}, R_{12}), (0, R_{12}), (a^{2n+1}, R_{34}), (0, R_{34}), syn),\\
R_{12} &= \{b : a^{2n+i} \to a^j \mid A_i \to bA_j \in R,\ i, j \in J\}\\
&\quad \cup \{b : a^{2n+i} \to a^{2n} \mid A_i \to b \in R,\ i \in J\},\\
R_{34} &= \{a^{2n+k} \to a^{2n} \mid k \in J\},\\
syn &= \{(1,2), (2,1), (1,4), (4,1), (2,3), (3,2), (3,4), (4,3)\}.
\end{aligned}$$

The system is also given in a graphical form in Figure 2. Note that it is finite and uses no forgetting rule.

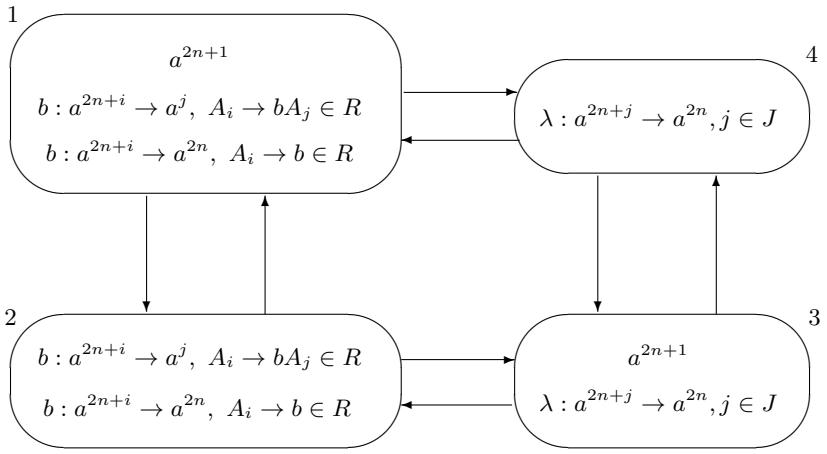

Fig. 2. An SN P system whose control language is a given regular language

In the first step, neurons 1 and 3 can fire; in the next step, neurons 2 and 4 fire – and the computation proceeds in steps which alternate the previous pairs of neurons. When a pair of neurons fires, then no spike remains inside these neurons, but the other pair receives spikes. This means that in each step a rule with a label $b \in T$ and one with the label λ are used (hence the computation is λ-label restricted).

With each nonterminal $A_i, 1 \leq i \leq n$, we have associated $2n+i$ spikes; initially, we have in neurons 1 and 3 spikes which identify the nonterminal $S = A_1$.

Assume that either σ_1, σ_3, or σ_2, σ_4 contain spikes, namely $2n+i$ in σ_1, σ_2, for a rule $A_i \to bA_j \in R$, and $2n+k$ in σ_3, σ_4, for some $k \in J$. The rule $A_i \to bA_j$ is simulated by using the rule (with label b) $a^{2n+i} \to a^j$ in σ_1, σ_2, simultaneously with using the rule (with label λ) $a^{2n+k} \to a^{2n}$ in σ_3, σ_4. The symbol b is added to the control word, and the process is continued with the simulation of a rule $A_j \to u \in R, u \in T \cup TN$.

In the moment when a (terminal) rule $A_i \to b \in R$ is simulated, the active σ_1 or σ_2 introduces $2n$ spikes, at the same time with $2n$ spikes produced by the paired neuron σ_3, σ_4. Two neurons are empty, the other two contains $4n$ spikes, hence no rule can be applied in any neuron. The computation halts, having as its control word the word generated by the derivation in G. Consequently, $Sz(\Pi) = L(G)$, which concludes the proof. □

We do not know whether the number of neurons in the previous theorem can be decreased.

7 Controlled P Systems

In the previous sections, the control words were collected in order to have a new way of producing a language starting from a P system. The computations cannot proceed freely, but they should be label restricted or λ-label restricted. This restriction has an influence on the computing power of P systems, considered as number computing devices. Indeed, let us consider the following systems:

$$\Pi_1 = (\{a, b\}, [\ ]_1, a, \{r_1 : a \to aa,\ r_2 : a \to b\}, 1),$$
$$\Pi_2 = (\{a, b\}, [\ ]_1, a, \{a, b\}, \{r_1 : (a, out; aa, in),\ r_2 : (a, out; b, in)\}, 1).$$

When only label restricted transitions are allowed, the two rules of each system cannot be used at the same time, hence we obtain $N_{lr}(\Pi_1) = N_{lr}(\Pi_2) = \{2^n \mid n \geq 0\}$ (we have added the subscript lr in order to indicate that only label restricted computations are allowed). This set of numbers cannot be generated by non-cooperative transition P systems, neither by symport-antiport P systems of this complexity (one membrane, two rules) with non-restricted computations.

A more general case is the one when a pair (Π, C) is considered, where Π is a P system of any type, with the rules labeled by elements of an alphabet H and $C \subseteq H^*$ is a given language. This language is used in order to restrict the computations in Π: only label restricted computations are allowed whose control words are in C. (This corresponds to controlled context-free grammars in regulated rewriting.)

The study of controlled P systems remains to be done (combining classes of P systems with types of control languages, as already done for Chomsky controlled

grammars). It is expected that a control language provides a powerful way to "program" the work of a P system.

8 Final Remarks

Many research topics were mentioned in the previous sections, many others remain to be explored. For instance, we have said nothing about tissue-like P systems – is anything interesting in this case from the language computing point of view? How this case compares with the four types of P systems considered above?

Another direction of investigation concerns sets of infinite sequences (also called ω-languages). Some results were reported in [15] for symport-antiport P systems, and in [29] and [14] for SN P systems.

A related issue was considered in [35]: handling languages over infinite alphabets.

Besides the previous ways to associate a language with a P system, also are other ideas were preliminarily explored. One of them is to encode a string in the membrane structure itself, and then handling the membrane structure means processing the string; see [5] for some details.

For all families of languages which are not equal to RE it makes sense to consider the classic problems investigated in formal language theory: closure properties, decidability, representation theorems, semilinearity, and so on. Also, the membership complexity is of interest (an issue considered already in [2]). In view of possible applications in modeling aspects related to natural languages, it would be of interest to find ways to generate *mildly context-sensitive languages* (semilinear, parsable in polynomial time, powerful enough to cover some non-context-free constructions in natural languages).

A related research direction concerns the translation of languages. Some attempts were reported in [11] and [25].

We can conclude with the observation that many things were done in membrane computing in handling languages by means of P systems with symbol objects, but a lot of work still remains to be carried out

Acknowledgements. Work supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. A. Alhazov, E. Boian, S. Cojocaru, Yu. Rogozhin: Modelling inflections in Romanian language by P systems with string replication. *Pre-proc. Tenth Workshop on Membrane Computing, WMC10*, Curtea de Argeş, August 2009 (M.J. Pérez-Jiménez, A. Riscos-Núñez, eds.), 116–128.
2. A. Alhazov, C. Ciubotaru, S. Ivanov, Yu. Rogozhin: Membrane systems languages are polynomial-time parsable. *Computer Science Journal of Moldova*, 18, 2 (2010), 139–148.

3. A. Alhazov, C. Ciubotaru, S. Ivanov, Y. Rogozhin: The family of languages generated by non-cooperative membrane systems. *Membrane Computing. 11th International Conference, CMC11, Jena, Germany, August 24-27, 2010. Revised, Selected, and Invited Papers* (M. Gheorghe et al., eds.), LNCS 6501, Springer, Berlin, 2010, 65–79.
4. A. Alhazov, C. Ciubotaru, Yu. Rogozhin, S. Ivanov: The membrane systems language class. *Proc. Eighth Brainstorming Week on Membrane Computing, Sevilla, 2010*, 23–35, and *Proc. LA Symposium*, RIMS Kôkyûroku Series 1691, Kyoto University, 2010, 44–50.
5. F. Bernardini, M. Gheorghe: Language generating by means of P systems with active membranes. *Proc. Brainstorming Week on Membrane Computing*, Technical Report, 26/03, Rovira i Virgili University, Tarragona, 2003, 46–60.
6. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75, 1-4 (2007), 141–162.
7. H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by spiking neural P systems. *Proc. Fourth Brainstorming Week on Membrane Computing*, Sevilla, 2006.
8. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In *Proc. Fourth Brainstorming Week on Membrane Computing*, Sevilla, 2006, RGNC Report 02/2006, 241–265.
9. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Handling languages with spiking neural P systems with extended rules. *Romanian J. Information Sci. and Technology*, 9, 3 (2006), 151–162.
10. G. Ciobanu, Gh. Păun, Gh. Ştefănescu: Sevilla carpets associated with P systems. *Proc. Brainstorming Week on Membrane Computing* (M. Cavaliere et al., eds.), Tarragona Univ., TR 26/03, 2003, 135–140.
11. G. Ciobanu, Gh. Păun, Gh. Ştefănescu: P transducers. *New Generation Computing*, 24, 1 (2006), 1–28.
12. E. Csuhaj-Varjú, Gy. Vaszil: P automata or purely communicating accepting P systems. *Membrane Computing, International Workshop, WMC-CdeA, Curtea de Argeş, Romania, August 19-23, 2002, Revised Papers* (Gh. Păun et al., eds.), LNCS 2597, Springer, 2003, 219–233.
13. R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez: On the power of P and dP automata. *Ann. Univ. Buc. Mathem.-Informatics Series*, 63 (2009), 5–22.
14. R. Freund, M. Oswald. Regular omega-languages defined by finite extended spiking neural P systems. *Fundamenta Informaticae*, 83 (2008), 65-73.
15. R. Freund, M. Oswald, L. Staiger. Omega-P automata with communication rules. *Pre-proc. Workshop on Membrane Computing, Tarragona, July 2003* (A. Alhazov et al., eds.), 252–265.
16. O.H. Ibarra, Gh. Păun: Characterizations of context-sensitive languages and other language classes in terms of symport/antiport P systems. *Theoretical Computer Sci.*, 358, 1 (2006), 88–103.
17. M. Ionescu: *Membrane Computing. Traces, Neural Inspired Models, Controls.* PhD Thesis, URV Tarragona, 2008.
18. M. Ionescu, C. Martin-Vide, Gh. Păun: P systems with symport/antiport rules: The traces of objects. *Grammars*, 5 (2002), 65–79.
19. M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez, T. Yokomori: Spiking neural dP systems. *Fundamenta Informaticae*, 11, 4 (2011), 423–436.

20. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71 (2006), 279–308.
21. M. Oswald: *P Automata*, PhD Thesis, TU Viena, 2003.
22. Gh. Păun: Computing with membranes. *J. Comput. Syst. Sci.*, 61 (2000), 108–143 (see also TUCS Report 208, November 1998 (`www.tucs.fi`).
23. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
24. Gh. Păun: Languages in membrane computing. Some details for spiking neural P systems. *Proc. 10th DLT Conf., Santa Barbara, USA, 2006*, LNCS 4036, Springer, Berlin, 2006, 20–35.
25. Gh. Păun: Spiking neural P systems used as acceptors and transducers. *Proc. CIAA 2007, 12th Conf.*, Prague, July 2007, LNCS 4783 (J. Holub, J. Zdarek, eds.), Springer, Berlin, 2007, 1–4.
26. Gh. Păun, M.J. Pérez-Jiménez: Solving problems in a distributed way in membrane computing: dP systems, *Int. J. of Computers, Communication and Control*, 5, 2 (2010), 238–252.
27. Gh. Păun, M.J. Pérez-Jiménez: P and dP automata: A survey. *Rainbow of Computer Science* (C.S. Calude, G. Rozenberg, A. Salomaa, eds.), LNCS 6570, Springer, Berlin, 2011, 102–115.
28. Gh. Păun, M.J. Pérez-Jiménez: An infinite hierarchy of languages defined by dP systems. *Theoretical Computer Sci.*, 431 (2012), 4–12.
29. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002.
30. Gh. Păun, G. Rozenberg, A. Salomaa: Membrane computing with an external output. *Fundamenta Informaticae*, 41, 3 (2000), 313–340
31. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
32. M.J. Pérez–Jiménez: A computational complexity theory in membrane computing. *Membrane Computing, Tenth International Workshop, WMC 2009, Curtea de Argeş Romania, August 2009, Selected and Invited Papers* (Gh. Păun et al., eds.), LNCS 5937, Springer, Berlin, 2009, 125–148.
33. A. Ramanujan, K. Krithivasan: Control words of spiking neural P systems. Paper in preparation, 2012.
34. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. 3 volumes, Springer, Berlin, 1998.
35. G. Vaszil: On a class of P automata as a machine model for languages over infinite alphabets. *Proc. Third Brainstorming Week on Membrane Computing, Sevilla, 2005* (M.A. Gutiérez-Naranjo et al., eds.), 317–325.
36. G. Vaszil: A class of P automata for characterizing context-free languages. *Proc. Fourth Brainstorming Week on Membrane Computing, Sevilla, 2006*, vol. II (C. Graciani et al., eds.), 267–276.
37. G. Vaszil: Variants of distributed P automata and the efficient parallelizability of langauges. *Membrane Computing. 12th Intern. Conf., CMC 2011, Fontainebleau, France, August 2011, Revised Selected Papers* (M. Gheorghe et al., eds.), LNCS 7184, Springer, Berlin, 2012, 51–61.
38. The P Systems Website: `http://ppage.psystems.eu`.